In [2]:

```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import didactic_datamining as ddm

from scipy.stats import mode
```

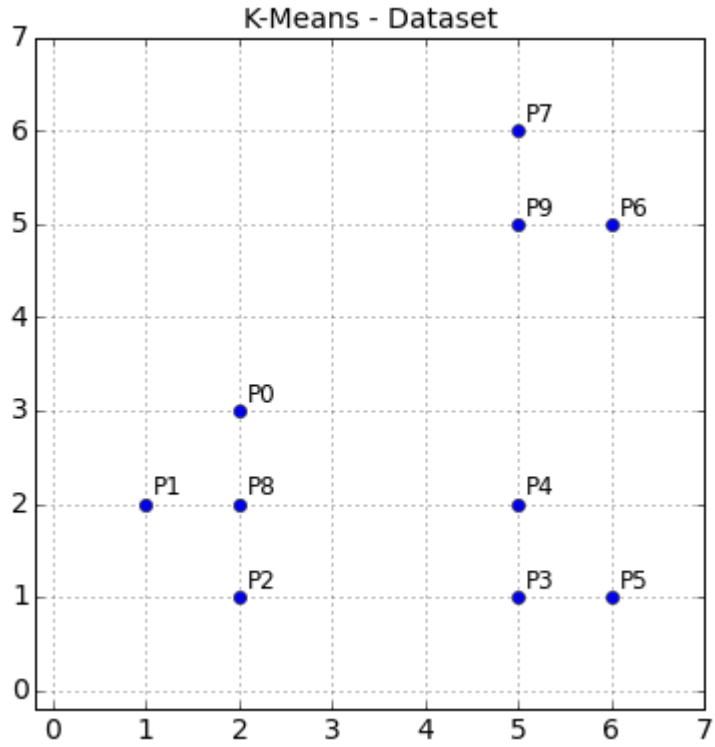
In [12]:

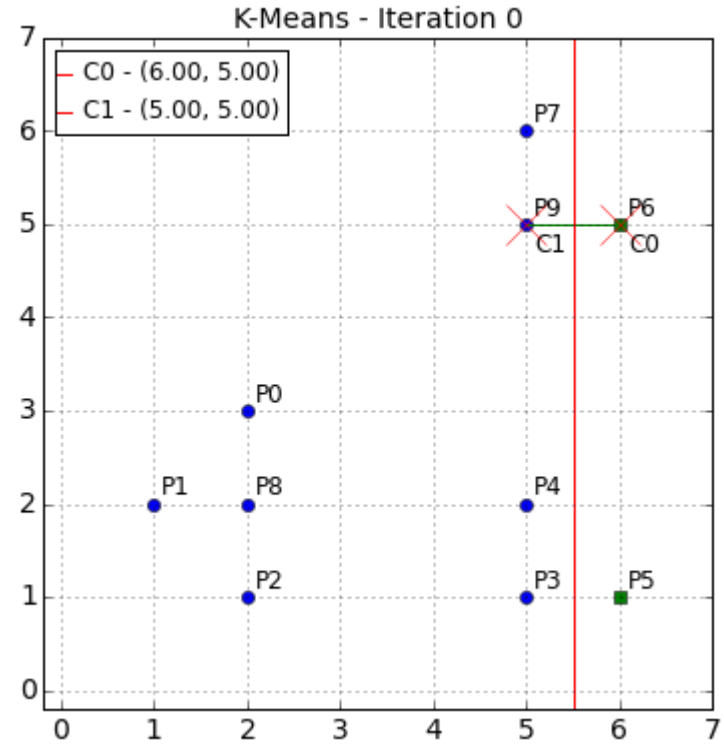
```
dataset = [[2, 3],
           [1, 2],
           [2, 1],
           [5, 1],
           [5, 2],
           [6, 1],
           [6, 5],
           [5, 6],
           [2, 2],
           [5, 5]] # originale esame

for row in dataset:
    print row[1] # '%s,%s' % (row[0], row[1])
```

```
3
2
1
1
2
1
5
6
2
5
```

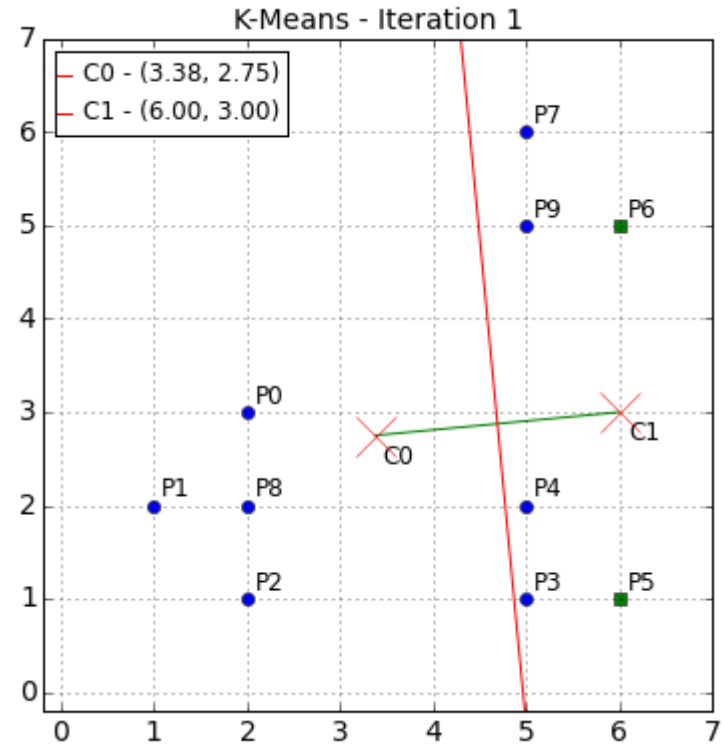
In [17]:

```
kmeans = ddm.DidatticKMeans(K=2, centroid_indexs=(6, 9), dist=ddm.euclidean_dist
ance) # 3,4 soluzione diversa
kmeans.fit(dataset, step_by_step=False)
```

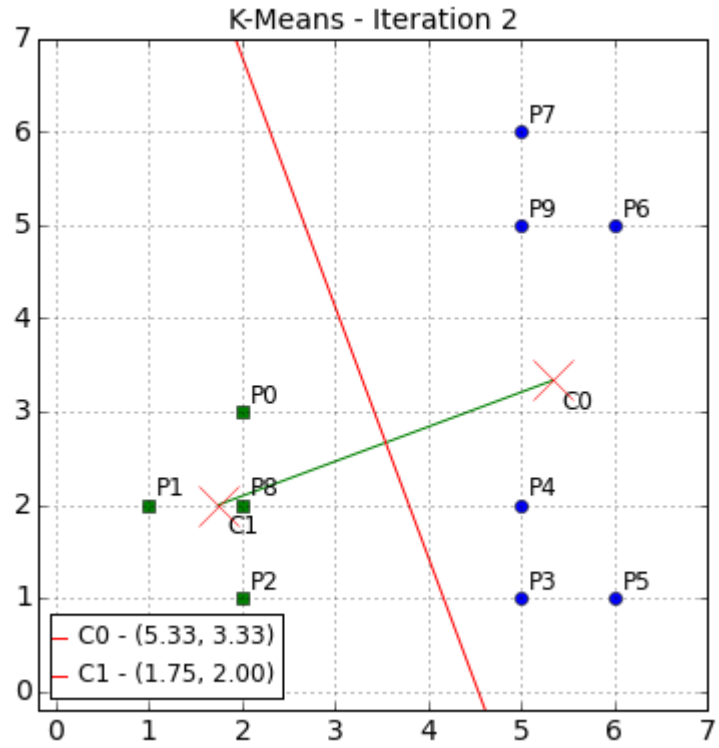

K-Means - Dataset
P7
P9
P6
P0
P1
P8
P4
P2
P3
P5
0
1
2
3
4
5
6
7



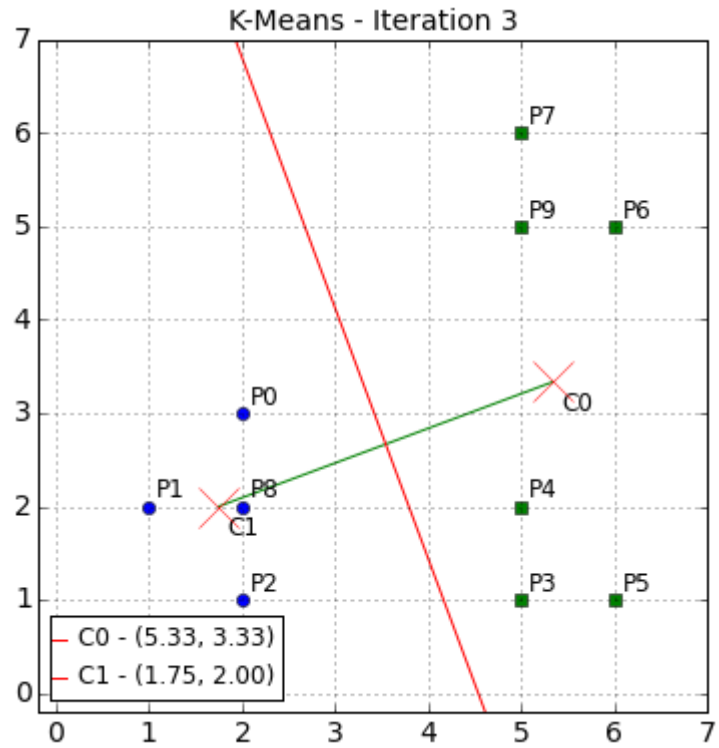

```
C0 (6.00, 5.00)
C1 (5.00, 5.00)
```



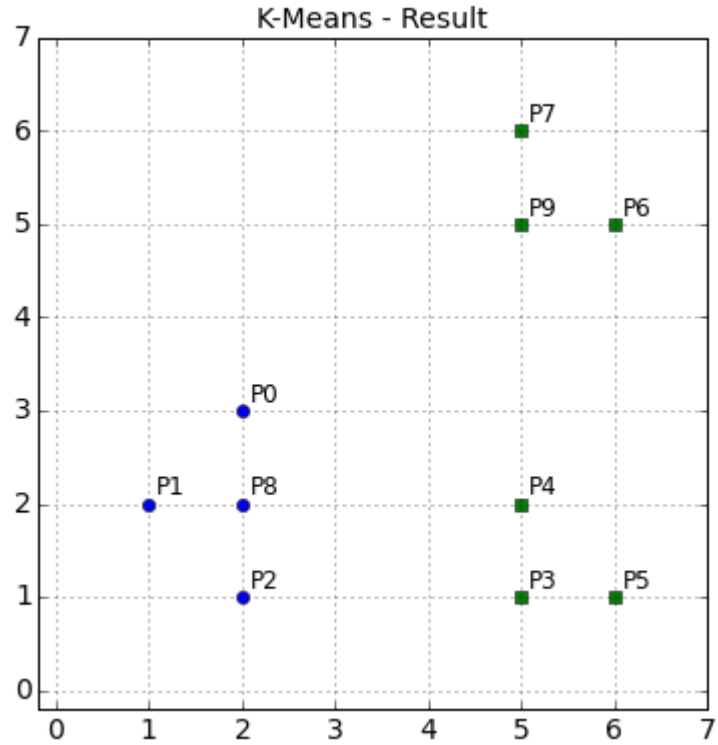
```
C0 (3.38, 2.75)
C1 (6.00, 3.00)
```




```
C0 (5.33, 3.33)
C1 (1.75, 2.00)
```




```
C0 (5.33, 3.33)
C1 (1.75, 2.00)
```



Out[17]:

```
<didactic_kmeans.DidatticKMeans instance at 0x1092b28c0>
```

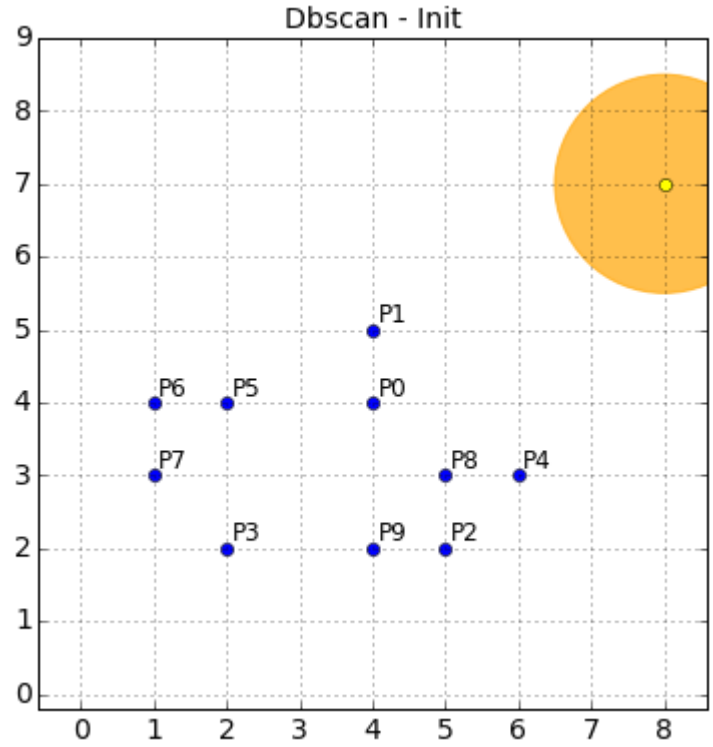
In [26]:

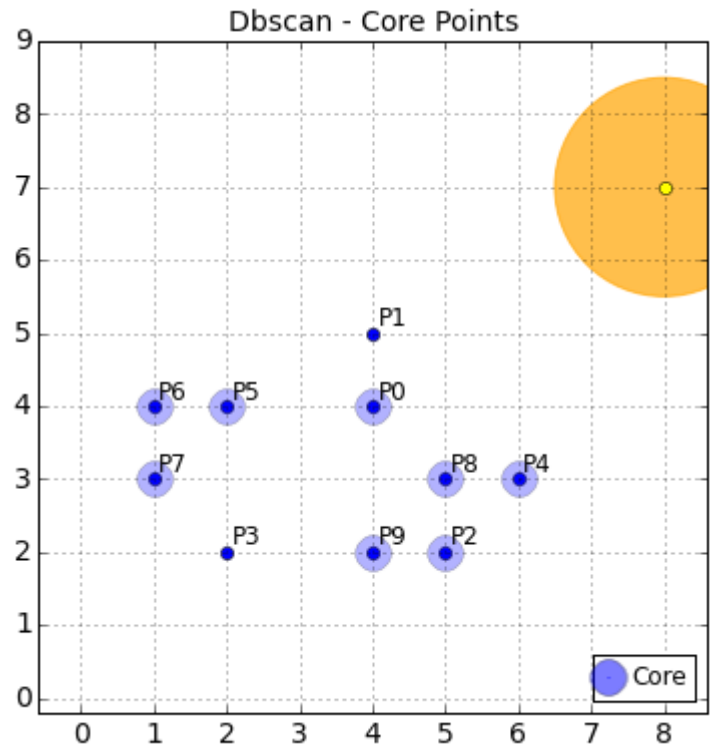
```
dataset = [[4, 4],
           [4, 5],
           [5, 2],
           [2, 2],
           [6, 3],
           [2, 4],
           [1, 4],
           [1, 3],
           [5, 3],
           [4, 2]
           ]

for row in dataset:
    print row[1]
```

```
4
5
2
2
3
4
4
3
3
2
```

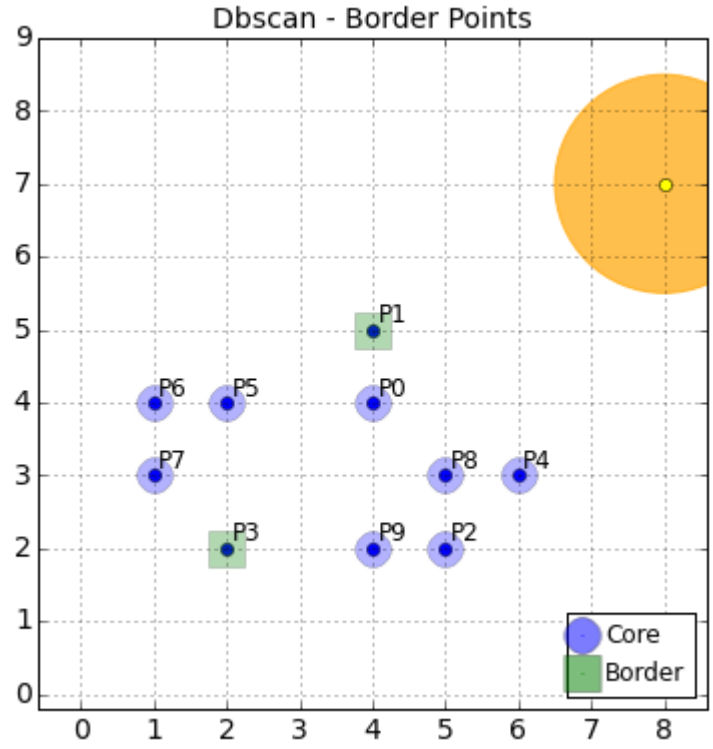
In [27]:

```
dbscan = ddm.DidatticDbscan(eps=1.5, min_pts=3)
dbscan.fit(dataset, step_by_step=False)
```

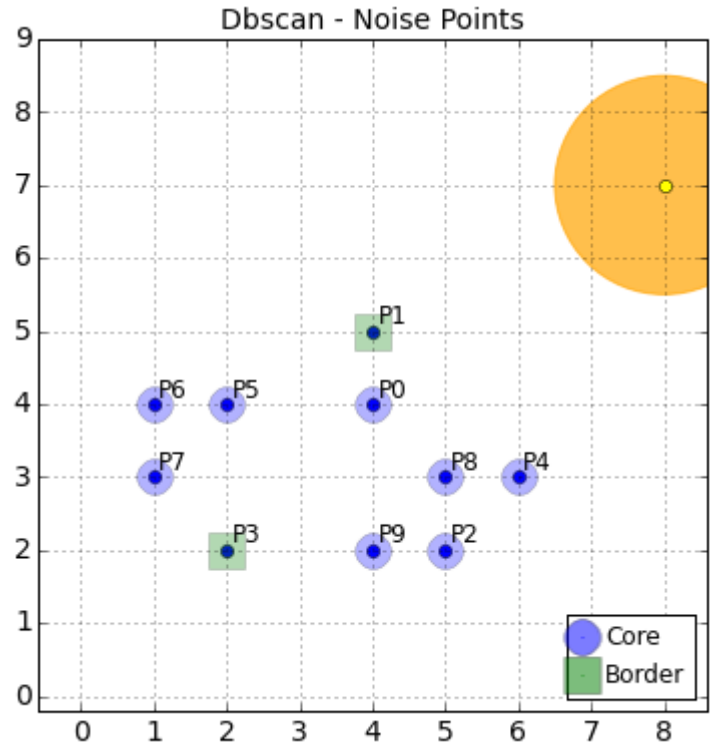




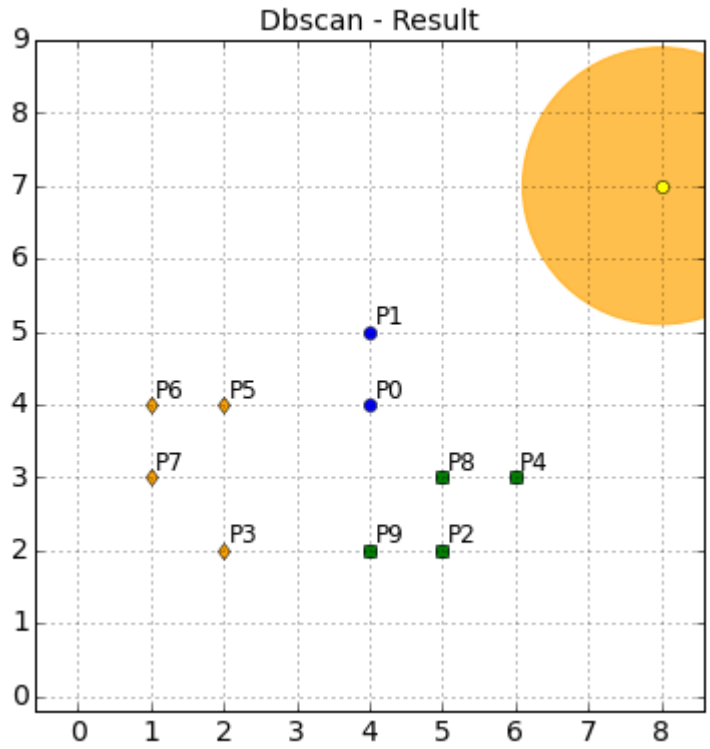

```
Core Points [0, 2, 4, 5, 6, 7, 8, 9]
```




```
Border Points [1, 3]
```



```
Noise Points []
```




```
defaultdict(<type 'set'>, {0: set([0, 1, 2, 4, 8, 9]), 1: set([8, 9,
2, 4]), 2: set([3, 5, 6, 7])})
```

In [39]:

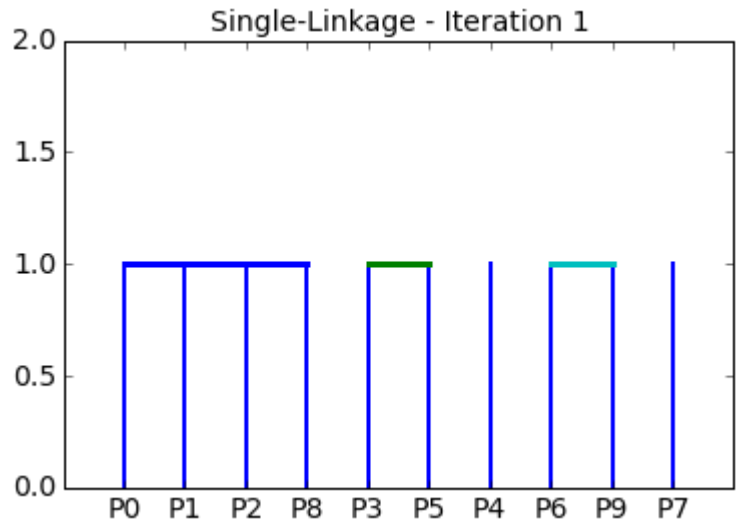
```
dm = [[ 0.,  2.,  2.,  5.,  3.,  6.,  6.,  4.,  1.,  5.],
 [ 2.,  0.,  2.,  5.,  5.,  6.,  8.,  6.,  1.,  7.],
 [ 2.,  2.,  0.,  3.,  5.,  4.,  8.,  6.,  1.,  7.],
 [ 5.,  5.,  3.,  0.,  2.,  1.,  5.,  7.,  4.,  4.],
 [ 3.,  5.,  5.,  2.,  0.,  3.,  3.,  5.,  4.,  2.],
 [ 6.,  6.,  4.,  1.,  3.,  0.,  4.,  8.,  5.,  5.],
 [ 6.,  8.,  8.,  5.,  3.,  4.,  0.,  4.,  7.,  1.],
 [ 4.,  6.,  6.,  7.,  5.,  8.,  4.,  0.,  5.,  3.],
 [ 1.,  1.,  1.,  4.,  4.,  5.,  7.,  5.,  0.,  6.],
 [ 5.,  7.,  7.,  4.,  2.,  5.,  1.,  3.,  6.,  0.]]

print '      ------------------------------------------------------------'
print '      |', ' | '.join(['P%s ' % a for a in range(0, 10)]), '|'
print '------------------------------------------------------------------'
for i, row in enumerate(dm):
    print '|  P%s |' % i, ' | '.join([str(a) for a in row]), '|'
    print '------------------------------------------------------------------'
```

```
      ------------------------------------------------------------
      | P0  | P1  | P2  | P3  | P4  | P5  | P6  | P7  | P8  | P9  |
------------------------------------------------------------------
|  P0 | 0.0 | 2.0 | 2.0 | 5.0 | 3.0 | 6.0 | 6.0 | 4.0 | 1.0 | 5.0 |
------------------------------------------------------------------
|  P1 | 2.0 | 0.0 | 2.0 | 5.0 | 5.0 | 6.0 | 8.0 | 6.0 | 1.0 | 7.0 |
------------------------------------------------------------------
|  P2 | 2.0 | 2.0 | 0.0 | 3.0 | 5.0 | 4.0 | 8.0 | 6.0 | 1.0 | 7.0 |
------------------------------------------------------------------
|  P3 | 5.0 | 5.0 | 3.0 | 0.0 | 2.0 | 1.0 | 5.0 | 7.0 | 4.0 | 4.0 |
------------------------------------------------------------------
|  P4 | 3.0 | 5.0 | 5.0 | 2.0 | 0.0 | 3.0 | 3.0 | 5.0 | 4.0 | 2.0 |
------------------------------------------------------------------
|  P5 | 6.0 | 6.0 | 4.0 | 1.0 | 3.0 | 0.0 | 4.0 | 8.0 | 5.0 | 5.0 |
------------------------------------------------------------------
|  P6 | 6.0 | 8.0 | 8.0 | 5.0 | 3.0 | 4.0 | 0.0 | 4.0 | 7.0 | 1.0 |
------------------------------------------------------------------
|  P7 | 4.0 | 6.0 | 6.0 | 7.0 | 5.0 | 8.0 | 4.0 | 0.0 | 5.0 | 3.0 |
------------------------------------------------------------------
|  P8 | 1.0 | 1.0 | 1.0 | 4.0 | 4.0 | 5.0 | 7.0 | 5.0 | 0.0 | 6.0 |
------------------------------------------------------------------
|  P9 | 5.0 | 7.0 | 7.0 | 4.0 | 2.0 | 5.0 | 1.0 | 3.0 | 6.0 | 0.0 |
------------------------------------------------------------------
```

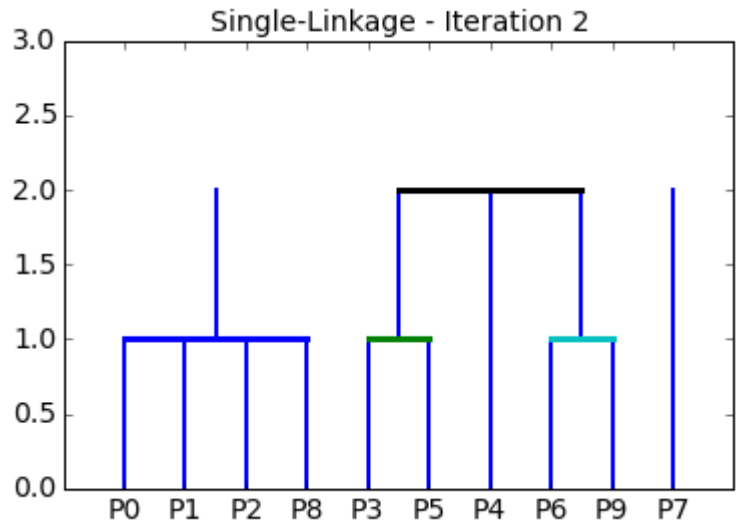
In [33]:

```
dataset = [[2, 3],
           [1, 2],
           [2, 1],
           [5, 1],
           [5, 3],
           [6, 1],
           [6, 5],
           [3, 6],
           [2, 2],
           [5, 5]]

hier = ddm.DidatticHierarchical()
hier.fit(dataset, link_criteria='min', use_distances=True, step_by_step=False, d
istance_type='cityblock')
```

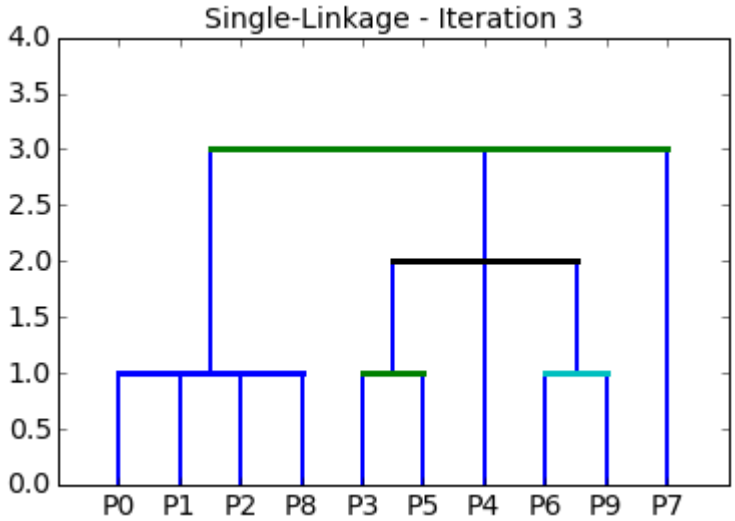
```
iter 0
distance merge 0.00
[(0,), (1,), (2,), (3,), (4,), (5,), (6,), (7,), (8,), (9,)]
[[ 0.  2.  2.  5.  3.  6.  6.  4.  1.  5.]
 [ 2.  0.  2.  5.  5.  6.  8.  6.  1.  7.]
 [ 2.  2.  0.  3.  5.  4.  8.  6.  1.  7.]
 [ 5.  5.  3.  0.  2.  1.  5.  7.  4.  4.]
 [ 3.  5.  5.  2.  0.  3.  3.  5.  4.  2.]
 [ 6.  6.  4.  1.  3.  0.  4.  8.  5.  5.]
 [ 6.  8.  8.  5.  3.  4.  0.  4.  7.  1.]
 [ 4.  6.  6.  7.  5.  8.  4.  0.  5.  3.]
 [ 1.  1.  1.  4.  4.  5.  7.  5.  0.  6.]
 [ 5.  7.  7.  4.  2.  5.  1.  3.  6.  0.]]
defaultdict(<type 'list'>, {0: [8], 1: [8], 2: [8], 3: [5], 5: [3],
6: [9], 8: [0, 1, 2], 9: [6]})
```




```
iter 1
distance merge 1.00
[(0, 1, 2, 8), (3, 5), (4,), (6, 9), (7,)]
[[ 0.  3.  3.  5.  4.]
 [ 3.  0.  2.  4.  7.]
 [ 3.  2.  0.  2.  5.]
 [ 5.  4.  2.  0.  3.]
 [ 4.  7.  5.  3.  0.]]
defaultdict(<type 'list'>, {1: [2], 2: [1, 3], 3: [2]})
```



```
iter 2
distance merge 2.00
[((0, 1, 2, 8),), ((3, 5), (4,), (6, 9)), ((7,),)]
[[ 0.  3.  4.]
 [ 3.  0.  3.]
 [ 4.  3.  0.]]
defaultdict(<type 'list'>, {0: [1], 1: [0, 2], 2: [1]})
```

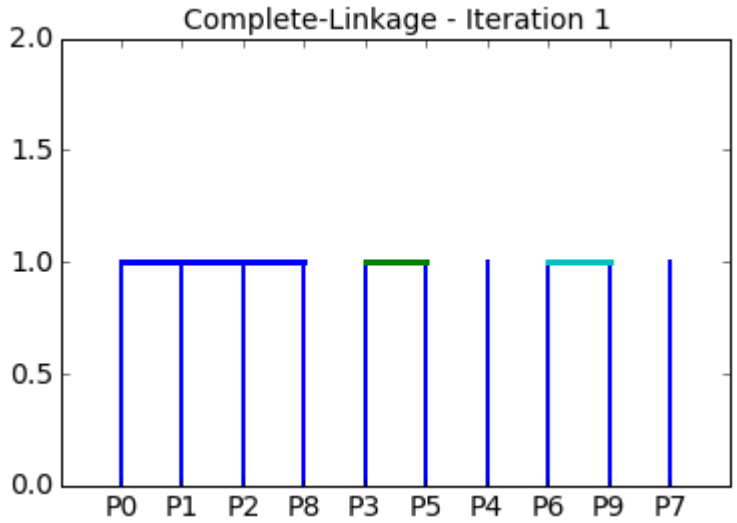



```
iter 3
distance merge 3.00
[(((0, 1, 2, 8),), ((3, 5), (4,), (6, 9)), ((7,),))]
[[ 0.]]
```

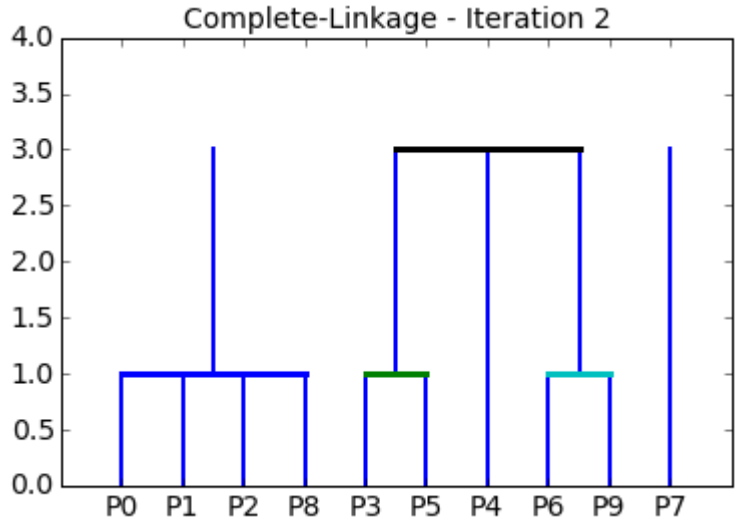
In [34]:

```
dataset = [[2, 3],
           [1, 2],
           [2, 1],
           [5, 1],
           [5, 3],
           [6, 1],
           [6, 5],
           [3, 6],
           [2, 2],
           [5, 5]]

hier = ddm.DidatticHierarchical()
hier.fit(dataset, link_criteria='max', use_distances=True, step_by_step=False, d
istance_type='cityblock')
```

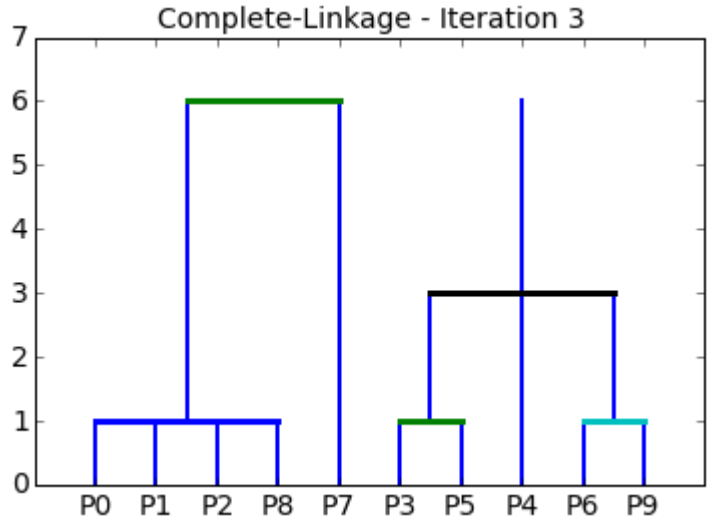
```
iter 0
distance merge 0.00
[(0,), (1,), (2,), (3,), (4,), (5,), (6,), (7,), (8,), (9,)]
[[ 0.  2.  2.  5.  3.  6.  6.  4.  1.  5.]
 [ 2.  0.  2.  5.  5.  6.  8.  6.  1.  7.]
 [ 2.  2.  0.  3.  5.  4.  8.  6.  1.  7.]
 [ 5.  5.  3.  0.  2.  1.  5.  7.  4.  4.]
 [ 3.  5.  5.  2.  0.  3.  3.  5.  4.  2.]
 [ 6.  6.  4.  1.  3.  0.  4.  8.  5.  5.]
 [ 6.  8.  8.  5.  3.  4.  0.  4.  7.  1.]
 [ 4.  6.  6.  7.  5.  8.  4.  0.  5.  3.]
 [ 1.  1.  1.  4.  4.  5.  7.  5.  0.  6.]
 [ 5.  7.  7.  4.  2.  5.  1.  3.  6.  0.]]
defaultdict(<type 'list'>, {0: [8], 1: [8], 2: [8], 3: [5], 5: [3],
6: [9], 8: [0, 1, 2], 9: [6]})
```



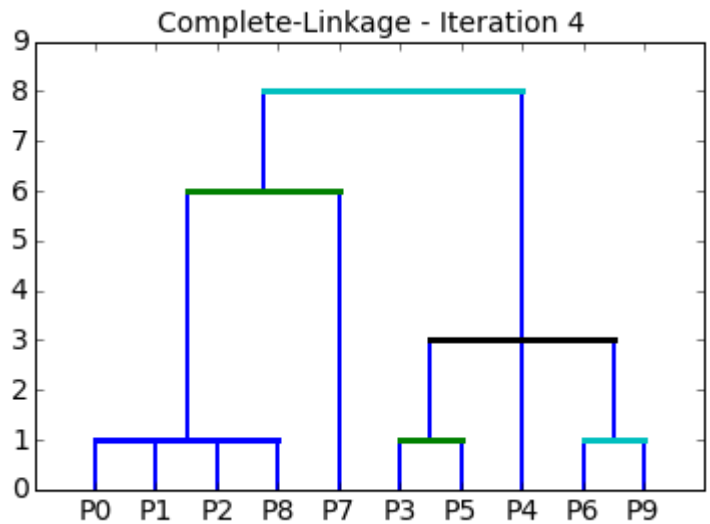

```
iter 1
distance merge 1.00
[(0, 1, 2, 8), (3, 5), (4,), (6, 9), (7,)]
[[ 0.  6.  5.  8.  6.]
 [ 6.  0.  3.  5.  8.]
 [ 5.  3.  0.  3.  5.]
 [ 8.  5.  3.  0.  4.]
 [ 6.  8.  5.  4.  0.]]
defaultdict(<type 'list'>, {1: [2], 2: [1, 3], 3: [2]})
```



```
iter 2
distance merge 3.00
[((0, 1, 2, 8),), ((3, 5), (4,), (6, 9)), ((7,),)]
[[ 0.  8.  6.]
 [ 8.  0.  8.]
 [ 6.  8.  0.]]
defaultdict(<type 'list'>, {0: [2], 2: [0]})
```




```
iter 3
distance merge 6.00
[(((0, 1, 2, 8),), ((7,),)), (((3, 5), (4,), (6, 9)),)]
[[ 0.  8.]
 [ 8.  0.]]
defaultdict(<type 'list'>, {0: [1], 1: [0]})
```




```
iter 4
distance merge 8.00
[((((0, 1, 2, 8),), ((7,),)), (((3, 5), (4,), (6, 9)),))]
[[ 0.]]
```

In [41]:

```
transactions = [
    ['B', 'A','R'],
    ['E', 'R'],
    ['B', 'E', 'R'],
    ['B', 'R', 'C'],
    ['A', 'B', 'E', 'F'],
    ['A', 'B', 'R'],
    ['C', 'R', 'F'],
    ['E', 'B', 'F', 'R'],
    ['A', 'R', 'F'],
    ['A', 'R', 'C', 'F'],
    ['E', 'B', 'F', 'A'],
]

apriori = ddm.DidatticApriori(min_sup=0.3, sup_type='r')
apriori.fit(transactions, step_by_step=False)
```

```
Apriori - Iteration 1
('A',) 0.55
('B',) 0.64
('C',) 0.27
('E',) 0.45
('F',) 0.55
('R',) 0.82
Apriori - Iteration 2
('A', 'B') 0.36
('A', 'C') 0.09 X
('A', 'E') 0.18 X
('A', 'F') 0.36
('A', 'R') 0.36
('B', 'C') 0.09 X
('B', 'E') 0.36
('B', 'F') 0.27
('B', 'R') 0.45
('C', 'F') 0.18 X
('C', 'R') 0.27
('E', 'F') 0.27
('E', 'R') 0.27
('F', 'R') 0.36
Apriori - Iteration 3
('A', 'B', 'F') 0.18 X
('A', 'B', 'R') 0.18 X
('A', 'F', 'R') 0.18 X
('B', 'E', 'F') 0.27
('B', 'E', 'R') 0.18 X
('B', 'F', 'R') 0.09 X
('E', 'F', 'R') 0.09 X
Apriori - Iteration 4
```

Out[41]:

```
<didactic_apriori.DidatticApriori instance at 0x109954560>
```

In [42]:

```
apriori.extract_rules(min_conf=0.7)
```

```
('A',) --> ('B',) conf: 0.67 X
('A',) --> ('F',) conf: 0.67 X
('A',) --> ('R',) conf: 0.67 X
('B',) --> ('A',) conf: 0.57 X
('B',) --> ('E',) conf: 0.57 X
('B',) --> ('E', 'F') conf: 0.43 X
('B',) --> ('F',) conf: 0.43 X
('B',) --> ('R',) conf: 0.71  lift: 0.87
('B', 'E') --> ('F',) conf: 0.75  lift: 1.38
('B', 'F') --> ('E',) conf: 1.00  lift: 2.20
('C',) --> ('R',) conf: 1.00  lift: 1.22
('E',) --> ('B',) conf: 0.80  lift: 1.26
('E',) --> ('B', 'F') conf: 0.60 X
('E',) --> ('F',) conf: 0.60 X
('E',) --> ('R',) conf: 0.60 X
('E', 'F') --> ('B',) conf: 1.00  lift: 1.57
('F',) --> ('A',) conf: 0.67 X
('F',) --> ('B',) conf: 0.50 X
('F',) --> ('B', 'E') conf: 0.50 X
('F',) --> ('E',) conf: 0.50 X
('F',) --> ('R',) conf: 0.67 X
('R',) --> ('A',) conf: 0.44 X
('R',) --> ('B',) conf: 0.56 X
('R',) --> ('C',) conf: 0.33 X
('R',) --> ('E',) conf: 0.33 X
('R',) --> ('F',) conf: 0.44 X
```

In [49]:

```
dataset_df = pd.read_csv('dataset_compito_dm_201802.csv',
                         skipinitialspace=True, delimiter=',')

for row in dataset_df.values:
    print ','.join([str(a) for a in row])
```

```
Yes,1,No,Yes,Yes
Yes,2,No,Yes,Yes
Yes,1,Yes,Yes,Yes
Yes,1,Yes,No,Yes
Yes,1,No,No,Yes
No,2,No,Yes,Yes
No,2,No,No,No
No,2,Yes,No,No
Yes,3,Yes,Yes,No
Yes,3,Yes,Yes,No
No,1,No,Yes,No
Yes,4,No,Yes,No
```

In [50]:

```
dataset_df
```

Out[50]:

	AvailableAtPurchase	Quantity	MultipleOrder	PremiumCustomer	DeliveredInTime
0	Yes	1	No	Yes	Yes
1	Yes	2	No	Yes	Yes
2	Yes	1	Yes	Yes	Yes
3	Yes	1	Yes	No	Yes
4	Yes	1	No	No	Yes
5	No	2	No	Yes	Yes
6	No	2	No	No	No
7	No	2	Yes	No	No
8	Yes	3	Yes	Yes	No
9	Yes	3	Yes	Yes	No
10	No	1	No	Yes	No
11	Yes	4	No	Yes	No

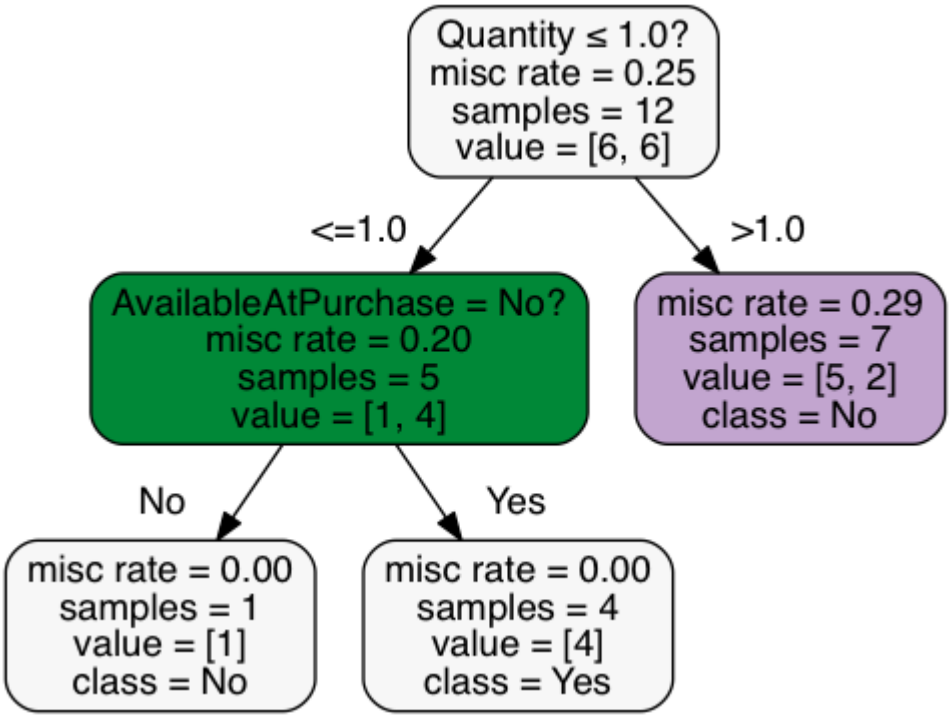
In [51]:

```
tree = ddm.DidatticClassificationTree(fun=ddm.error_rate, fun_name='misc rate',
                                      min_samples_split=2, min_samples_leaf=1, s
tep_by_step=False)
tree.fit(dataset_df, target='DeliveredInTime')
```

```
Root
Parent
        1 - 6/12 = 6/12
AvailableAtPurchase ['No', 'Yes']
        No 4
                No, 3/4
                Yes, 1/4
                1 - 3/4 = 1/4
        Yes 8
                No, 3/8
                Yes, 5/8
                1 - 5/8 = 3/8
        (3/8 * 8/12) + (1/4 * 4/12) = 4/12
        Delta Gain: 6/12 - 4/12 = 2/12

Quantity# ['<=1', '>1']
        <=1 5
                No, 1/5
                Yes, 4/5
                1 - 4/5 = 1/5
        >1 7
                No, 5/7
                Yes, 2/7
                1 - 5/7 = 2/7
        (1/5 * 5/12) + (2/7 * 7/12) = 3/12
        Delta Gain: 6/12 - 3/12 = 3/12

Quantity# ['<=2', '>2']
        <=2 9
                No, 3/9
                Yes, 6/9
                1 - 6/9 = 3/9
        >2 3
                No, 3/3
                1 - 3/3 = 0/3
        (3/9 * 9/12) + (0/3 * 3/12) = 3/12
        Delta Gain: 6/12 - 3/12 = 3/12

Quantity# ['<=3', '>3']
        <=3 11
                No, 5/11
                Yes, 6/11
                1 - 6/11 = 5/11
        >3 1
                No, 1/1
                1 - 1 = 0
        (0 * 1/12) + (5/11 * 11/12) = 5/12
        Delta Gain: 6/12 - 5/12 = 1/12

MultipleOrder ['No', 'Yes']
        No 7
                No, 3/7
                Yes, 4/7
                1 - 4/7 = 3/7
        Yes 5
                No, 3/5
                Yes, 2/5
                1 - 3/5 = 2/5
        (2/5 * 5/12) + (3/7 * 7/12) = 5/12
        Delta Gain: 6/12 - 5/12 = 1/12
```

```
PremiumCustomer ['No', 'Yes']
        No 4
                No, 2/4
                Yes, 2/4
                1 - 2/4 = 2/4
        Yes 8
                No, 4/8
                Yes, 4/8
                1 - 4/8 = 4/8
        (4/8 * 8/12) + (2/4 * 4/12) = 6/12
        Delta Gain: 6/12 - 6/12 = 0/12

--> Split By Quantity#1

Root_Quantity#1?<=1.0
Parent
        1 - 4/5 = 1/5
AvailableAtPurchase ['No', 'Yes']
        No 1
                No, 1/1
                1 - 1 = 0
        Yes 4
                Yes, 4/4
                1 - 4/4 = 0/4
        (0/4 * 4/5) + (0 * 1/5) = 0/5
        Delta Gain: 1/5 - 0/5 = 1/5

MultipleOrder ['No', 'Yes']
        No 3
                No, 1/3
                Yes, 2/3
                1 - 2/3 = 1/3
        Yes 2
                Yes, 2/2
                1 - 2/2 = 0/2
        (0/2 * 2/5) + (1/3 * 3/5) = 1/5
        Delta Gain: 1/5 - 1/5 = 0/5

PremiumCustomer ['No', 'Yes']
        No 2
                Yes, 2/2
                1 - 2/2 = 0/2
        Yes 3
                No, 1/3
                Yes, 2/3
                1 - 2/3 = 1/3
        (1/3 * 3/5) + (0/2 * 2/5) = 1/5
        Delta Gain: 1/5 - 1/5 = 0/5

--> Split By AvailableAtPurchase#No&Yes

Root_Quantity#1?<=1.0_AvailableAtPurchase#No&Yes?No
Parent
        1 - 1 = 0
--> No gain 1. Stop

Root_Quantity#1?<=1.0_AvailableAtPurchase#No&Yes?Yes
Parent
        1 - 4/4 = 0/4
--> No gain 1. Stop
```

```
Root_Quantity#1?>1.0
Parent
        1 - 5/7 = 2/7
AvailableAtPurchase ['No', 'Yes']
        No 3
                No, 2/3
                Yes, 1/3
                1 - 2/3 = 1/3
        Yes 4
                No, 3/4
                Yes, 1/4
                1 - 3/4 = 1/4
        (1/4 * 4/7) + (1/3 * 3/7) = 2/7
        Delta Gain: 2/7 - 2/7 = 0/7

Quantity# ['<=2', '>2']
        <=2 4
                No, 2/4
                Yes, 2/4
                1 - 2/4 = 2/4
        >2 3
                No, 3/3
                1 - 3/3 = 0/3
        (2/4 * 4/7) + (0/3 * 3/7) = 2/7
        Delta Gain: 2/7 - 2/7 = 0/7

Quantity# ['<=3', '>3']
        <=3 6
                No, 4/6
                Yes, 2/6
                1 - 4/6 = 2/6
        >3 1
                No, 1/1
                1 - 1 = 0
        (0 * 1/7) + (2/6 * 6/7) = 2/7
        Delta Gain: 2/7 - 2/7 = 0/7

MultipleOrder ['No', 'Yes']
        No 4
                No, 2/4
                Yes, 2/4
                1 - 2/4 = 2/4
        Yes 3
                No, 3/3
                1 - 3/3 = 0/3
        (0/3 * 3/7) + (2/4 * 4/7) = 2/7
        Delta Gain: 2/7 - 2/7 = 0/7

PremiumCustomer ['No', 'Yes']
        No 2
                No, 2/2
                1 - 2/2 = 0/2
        Yes 5
                No, 3/5
                Yes, 2/5
                1 - 3/5 = 2/5
        (2/5 * 5/7) + (0/2 * 2/7) = 2/7
        Delta Gain: 2/7 - 2/7 = 0/7

--> No gain 2. Stop
```



In [52]:

```
test_df = pd.read_csv('testset_compito_dm_201802.csv', skipinitialspace=True, de
limiter=',')

for row in test_df.values:
    print ','.join([str(a) for a in row])
```

```
Yes,1,No,Yes,Yes
Yes,2,No,No,No
No,1,Yes,Yes,Yes
Yes,3,Yes,Yes,No
Yes,1,No,No,Yes
No,2,No,Yes,No
```

In [53]:

```
prediction = tree.predict(test_df)
test_df['Predicted'] = prediction
for row in test_df.values:
    print ','.join([str(a) for a in row])
```

```
Yes,1,No,Yes,Yes,Yes
Yes,2,No,No,No,No
No,1,Yes,Yes,Yes,No
Yes,3,Yes,Yes,No,No
Yes,1,No,No,Yes,Yes
No,2,No,Yes,No,No
```

In [54]:

```
tree.evaluate(test_df)
```

```
R\P     |No     |Yes    |
No      |3      |0      |
Yes     |1      |2      |
Precision 1 1.0
Recall 2/3 0.666666666667
F1-measure 4/5 0.8
Accuracy 5/6 0.833333333333
```

```
/Library/Python/2.7/site-packages/numpy-override/numpy/core/fromnume
ric.py:2507: VisibleDeprecationWarning: `rank` is deprecated; use th
e `ndim` attribute or function instead. To find the rank of a matrix
see `numpy.linalg.matrix_rank`.
  VisibleDeprecationWarning)
```